Sampling Strategies for Enhanced Recommendation Performance: Advancements in LightGCN

Alexandru Turcu, Bogdan Palfi, and Ryan Amaudruz

University of Amsterdam
alexandru.turcu@student.uva.nl
bogdan.palfi@student.uva.nl
ryan.amaudruz@student.uva.nl

Abstract. Graph Convolutional Networks (GCNs) have seen increased popularity in the context of collaborative filtering based Recommender Systems. LightGCN is a heavily simplified variant of GCN, specifically designed for recommendation tasks. While at the time of publishing, LightGCN was able to achieve state-of-the-art performance, there was still room for improvement through the sampling strategy. This paper replicates the results of the original LightGCN paper and tests the model on a new dataset to further demonstrate its robustness. Furthermore, additional sampling methods are developed and tested on all datasets, some of which show increased performance in terms of recall or catalog coverage.

Keywords: Recommender Systems · LightGCN · GCN

1 Introduction

Due to increasing amount of online information, recent years have seen a substantial rise in the usage of recommender systems, which provide users with personalized recommendations. Such systems often rely on users' previous interactions with specific items such as movies, songs, or products to find other relevant items.

Collaborative Filtering (CF) [7] is a common technique used in recommender systems in which personalized recommendations are provided based on past interactions between users and items within the same community. CF is rooted in the idea that similar individuals have a high chance of benefiting from similar recommendations. CF can be further divided into two branches: memory-based, in which rating historical data is used to compute the similarity between users or items, and model-based, which focuses on machine learning algorithms to predict users' rating of unrated items.

Model-based approaches have been the focus of academic research in the last few years, Graph Convolutional Networks (GCNs) [5] being one of the most prominent additions to the field of CF. GCNs are a type of neural networks that

are designed to perform tasks such as regression and classifications on graph-structured data. They take advantage of the innate graph structure of certain datasets (such as social networks, knowledge graphs, the World Wide Web, etc.) to propagate information between the nodes of the graph. The intuition behind using GCNs for recommender systems is that user behavior can be modeled under a graph structure: users and items can be represented by nodes, while the interactions between them can be denoted by edges.

LightGCN [4] is one variant of GCNs that is fine-tuned for recommendations. The authors claim that by simplifying the architecture of a standard GCN, the model becomes more concise and appropriate for recommendation tasks. LightGCN achieved state-of-the-art results, outperforming other recommendation models at the time of publishing (2020).

Our work aims to replicate the main results of the LightGCN paper [4] and further demonstrate the algorithm's robustness by testing it on the LastFM [1] dataset. Secondly, as an extension, we attempt to improve the overall performance of the model by designing new data sampling strategies to be used during the training process.

2 Related Work

2.1 Graph Convolutional Networks

First introduced in [5], GCNs are a form of convolutional neural networks (CNN) that allow for semi-supervised learning directly from graph-structured data. Similar to classic CNN models, the main idea of GCNs is to extract features not only from a node but also from its neighbors. However, while CNN models function on grid-like, Euclidean data such as images, GCNs do not have this limitation and are suited for irregular, non-Euclidean data. As such, GCNs can be considered a more generalized version of CNNs, which can operate on various graph structures by taking into consideration the node connectivity in the graph. For this reason, GCNs have seen an increase in popularity in the context of recommender systems. Specifically, GCNs can use the information contained in user-item interaction graphs, which are highly irregular and sparse, to create user and item embeddings, which are then used when predicting the relevance of items to the user.

2.2 Neural Graph Collaborative Filtering

Neural Graph Collaborative Filtering (NGCF) [8] is a GCN-inspired model that calculates the user and item embeddings based on the user-item interaction graph, according to the following formulas:

$$\mathbf{e}_u^{(k+1)} = \sigma\left(\mathbf{W}_1\mathbf{e}_u^{(k)} + \sum_{i\in\mathcal{N}_u} \frac{1}{\sqrt{|\mathcal{N}_u||\mathcal{N}_i|}}(\mathbf{W}_1\mathbf{e}_i^{(k)} + \mathbf{W}_2(\mathbf{e}_i^{(k)} \odot \mathbf{e}_u^{(k)}))\right) \quad (1)$$

$$\mathbf{e}_i^{(k+1)} = \sigma\left(\mathbf{W}_1\mathbf{e}_i^{(k)} + \sum_{u\in\mathcal{N}_i}\frac{1}{\sqrt{|\mathcal{N}_u||\mathcal{N}_i|}}(\mathbf{W}_1\mathbf{e}_u^{(k)} + \mathbf{W}_2(\mathbf{e}_u^{(k)}\odot\mathbf{e}_i^{(k)}))\right) \quad (2)$$

In the given context, the embeddings are calculated over multiple layers, such that $\mathbf{e}_u^k$ and $\mathbf{e}_i^k$ represent the embedding of the user u and item i after k levels. Initially, $\mathbf{e}_u^0$ and $\mathbf{e}_u^0$ denote the ID embeddings of their respective user and item. σ stands for the nonlinear activation function, while $\mathcal{N}_u$ represents the set of items that the user has interacted with and, analogously, $\mathcal{N}_i$ is the set of users that have interacted with the item. The features of the previous embeddings are transformed with the use of the $\mathbf{W}_1$ and $\mathbf{W}_2$ trainable weight matrices. The final embeddings are obtained by concatenating the embeddings at each of the L layers, while the prediction score for each possible recommendation is obtained by calculating the inner product between the user and item embeddings.

2.3 LightGCN

LightGCN [4] is a simplified version of NGCF in which the feature transformation and nonlinear activation parts from Equations 1 and 2 are removed. In this way, only the neighborhood aggregation component of GCNs is leveraged. The main idea behind this simplified model comes from the fact that nodes in user-item interaction graphs simply denote one-hot encodings of users or items, thus not containing any semantic information. Because of this, [4] argue that applying feature transformation and nonlinear activation on the nodes is not only futile but also counterproductive since the model would have to learn more trainable parameters. Therefore, the user and item embedding Equations 1 and 2 from the NGCF model are simplified as seen in Equations 3 and 4, respectively.

$$\mathbf{e}_u^{(k+1)} = \sum_{i\in\mathcal{N}_u}\frac{1}{\sqrt{|\mathcal{N}_u||\mathcal{N}_i|}}\mathbf{e}_i^{(k)} \quad (3)$$

$$\mathbf{e}_i^{(k+1)} = \sum_{u\in\mathcal{N}_i}\frac{1}{\sqrt{|\mathcal{N}_u||\mathcal{N}_i|}}\mathbf{e}_u^{(k)} \quad (4)$$

Additionally, compared to NGCF, LightGCN does not directly employ self-connections in the graph but instead uses layer combination to achieve the same results [4]. Specifically, the user and item embeddings at each layer k are combined according to Equation 5:

$$\mathbf{e}_u = \sum_{k=0}^{K}\alpha_k\mathbf{e}_u^{(k)}; \qquad \mathbf{e}_i = \sum_{k=0}^{K}\alpha_k\mathbf{e}_i^{(k)}; \quad (5)$$

where $\alpha_k \geq 0$ acts as a weight for each layer k and is set to a constant value of $\frac{1}{K+1}$. In this way, LightGCN only learns $e_u^{(0)}$ and $e_i^{(0)}$, the user and item embeddings at layer 0.

3 Methodology

3.1 Reproducing LightGCN

To ensure a fair comparison between the original results from the LightGCN paper [4] and the extensions presented in this work, the first step was to reproduce the main claims of [4]. The focus was on reproducing Table 3, which we consider to be the main contribution of the authors and from which the original NGCF and LightGCN results were taken. Therefore, the LightGCN model, for which the code was available online [1], was trained and tested under 4 configurations and 3 datasets. Specifically, as in [4], we used 1-4 layers and tested on the Gowalla, Yelp2018 and Amazon-Book datasets (see Section 3.3 for information on the datasets).

3.2 Sampling methods

The main contribution of this work comes in the form of analyzing the performance of different sampling methods used during the training of the LightGCN model. The sampling methods were developed by analyzing their performance on the Gowalla dataset and tested on the remaining datasets to assess their generalization capabilities. Below, we review the sampling method used in the original work and introduce the new sampling strategies.

Original sampling Although the original paper does not explicitly state the sampling strategy employed, the codebase reveals the authors of the paper implement two different sampling methods: one in Python, with a runtime of approximately 9 seconds, and another written in C++, with a runtime below 0.1 seconds. The Python implementation uniformly samples users with replacement, obtaining n_{train} users. It then loops through the users and uniformly samples both a positive and negative item for each user. On the other hand, the C++ version derives the mean number of items per user and rounds the value to an integer to obtain μ_N. For each unique user, it then uniformly samples μ_N sets containing a positive and a negative item. For the purpose of this report, we will assume that the authors used the faster C++ and will refer to it as the original sampling method from now.

Given this sampling method, the sparse nature of user-item interaction graphs influences the probability that $Item_i$ will be selected. Specifically, due to the abundance of negative samples, the probability of an item being chosen as a negative item is similar for different candidates regardless of their popularity. On the other hand, the probability of an item being selected as a positive sample heavily depends on the item's popularity. Considering that the pool of positive items is limited, the probability of selecting positive samples is further destabilized, which could be considered a disadvantage of this sampling method.

[1] `https://github.com/gusye1234/LightGCN-PyTorch`

In Equation 6, we show the expected occurrences of $Item_i$ as a positive item in the training set:

$$\mathbb{E}[Item_i] = \mu_N \sum_{u \in \mathcal{U}_i} \frac{1}{|\mathcal{N}_u|} \tag{6}$$

In the equation above, μ_N is the mean number of items per user rounded to the closest integer, $\mathcal{U}_i$ is the set of users that have interacted with $Item_i$ and $\mathcal{N}_u$ is the set of items that the user u has interacted with. Table 1 shows the minimum and maximum value $\mathbb{E}[Item_i]$ found in the different datasets, from which it can be seen that items with high popularity amongst users (maximum column) will be sampled as positives significantly more than their less popular counterparts (minimum column). We hypothesize that this bias is non-optimal and introduce a sampling method that reduces this bias.

Dataset	**Minimum**	**Mean**	**Maximum**	**Std**
gowalla	0.1459	19.6717	1936.5017	37.2246
lastfm	0.0971	49.9498	4749.4860	96.6568
yelp2018	0.1696	32.4604	1577.5947	53.4341
amazon-book	0.0042	25.8620	2170.7666	43.9009

Table 1. Expected Item Appearance as Positive

Weighted item probability sampling Table 1 shows how the most popular items will appear significantly more (up to 10,000 times more) than their non-popular counterpart. While this bias is beneficial to a certain extent, as more popular items will be more likely to be appreciated by the average users, we argue that the high imbalance in the expected number of items occurring as positives can lead to a sub-optimal sampling strategy.

To understand the impact of this imbalance on the model performance, Figure 1 shows how precision and recall are influenced by item popularity on the Gowalla dataset. It can be seen that items with 20 or more user connections exhibit a recall of 25.2% and a precision of 5.6%. In contrast, less popular items with fewer than 10 user connections demonstrate a significantly lower recall of 1.0% but a higher precision. With a precision of 8.1%, unpopular items suggested to users were about 45% more likely to be correct. As a result, we hypothesize that increasing the sampling probability of less popular items would result in improved performance. In this way, the number of sampled popular items with low precision would be reduced while the number of unpopular items with high precision would increase. Specifically, we propose a strategy that increases the sampling likelihood of items that are below the median number of users $Mdn[Item_i]$, using Equation 7 to derive their adjusted probability weight.

$$\mathbb{P}[Item_i]_{actual} = \mathbb{P}[Item_i]_{original} \sqrt{Mdn[Item_i]/|\mathcal{N}_i|} \tag{7}$$

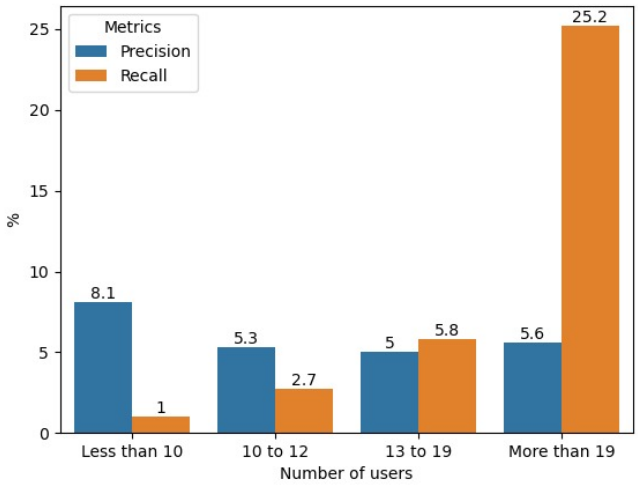

Fig. 1. Item precision and recall by number of user connections - Gowalla dataset

Hard negative sampling The rationale behind the hard negative sampling strategy has its origin in the information retrieval domain. In the case of contrastive learning, it is advantageous for the training process if the negative samples are harder to discriminate. For instance, more information can be extracted from the difference between two texts on the same topic rather than two completely different topics. Training with the use of negative samples requires the model to develop a deep semantic understanding of the text rather than just the ability to discriminate topics. As LightGCN uses contrastive learning, it is expected that some negative samples are rather easy to discriminate and therefore offer less learning potential than other harder samples. As a result, the Hard Negative sampling strategy for LightGCN was derived with the same intention: providing harder examples to discriminate in order to improve the learning potential.

To understand the sampling strategy, the testing process must also be explained. At inference time, the inner product between the final embedding of each user-item pair is derived. For each user, the top-k items with the highest inner product are selected while also excluding the items that are known to be positive from the training set. Therefore, the negative samples that are harder to discriminate have a high inner product with respect to the user.

During the first 10 epochs of training, the hard negative sampling first employs the original random sampling strategy. Every 10 epochs, an inference is made to monitor the test metrics. During the inference, 13% of the hardest samples for each user are recorded. This value of 13% was found empirically. After the first 10 epochs, the method samples from the hard negative sample pool with

a probability of 1% or 5% for the low and high probability versions. In practice, storing 13% of the hardest samples is infeasible when it comes to memory requirements. Therefore, the method samples uniformly $\frac{3}{13}^{th}$ of the hard samples, such that the negative samples can be stored in memory. In turn, this allows for a larger sample pool which ensures diversity in the samples.

While there are many different strategies to find hard negative samples, the time constraint for this project did not allow for in-depth analysis and optimization of the strategy. For the purpose of this report, we chose a strategy that would increase the hardness of the samples such that we can determine if the prioritization of hard samples improves the model outcome.

Mixed sampling The mixed sampling strategy combines the weighted item probability and hard negative sampling strategies. The weighted item probability sampling adjusts the probability of items being selected as positive depending on the item popularity, while the hard negative sampling limits the pool of negative samples to select from. As both strategies act independently on different sample labels, they can be combined seamlessly and the potential benefit can be investigated.

3.3 Experimental Setup

Benchmark Datasets The datasets used for running the experiments are Gowalla [6], Amazon-Book [3] and Yelp2018 [4,8], which are the same as those used in the LightGCN paper [4]. An additional dataset, not included in the original paper, is LastFM [1], which was included to test the reliability of LightGCN on novel datasets. All four datasets are comprised of user-item interaction booleans, where the user has either interacted with the item or not. The dataset statistics are provided in Table 3.3.

Dataset	User #	Item #	Interaction #	Density
Gowalla	29,858	40,981	1,027,370	0.00084
Yelp2018	31,668	38,048	1,561,406	0.00130
Amazon-Book	52,643	91,599	2,984,108	0.00062
LastFM	23,566	48123	3,034,796	0.00081

Table 2. Statistics of the experimented data [4]

Hyperparameters and Training Procedure To ensure a fair assessment of the sampling methods, most parameters and hyperparameters were identical to those used in the LightGCN paper [4]. For instance, the batch size used during training was 2048 for the Amazon-Book dataset and 1024 for the rest of the datasets, including LastFM. The only difference between [4] and this work is the number of epochs used. Specifically, [4] stated that 1000 epochs

were used for all datasets and all configurations, which was also done in this work when reproducing the main claims of [4]. However, when evaluating the new sampling methods, the model was trained for 1500 epochs for the Gowalla dataset, 1000 epochs for the Yelp2018 and LastFM datasets, and 850 epochs for the Amazon-Book dataset. This was done to account for the dataset size and the computational and time limitations of this project. In addition, considering the same limitations, only the Gowalla dataset was employed to find the 2 best sampling methods under the 3-layer configuration. Once the 2 methods were selected, they were also tested on the other 3 datasets.

Catalog coverage metric Catalog coverage is a metric used to evaluate the extent to which recommended items cover the overall catalog, reflecting the breadth of selection available to users. Catalog coverage is defined in Equation 8:

$$coverage = \frac{|I_{recommended}|}{|I_{total}|} * 100 \tag{8}$$

where $|I_{recommended}|$ is the number of recommended user-item pairs and $|I_{total}|$ is the total number of items in the catalog. This additional metric was introduced to further assess the differences between sampling methods, under the assumption that a higher catalog coverage should lead to a more thorough exploration of possible recommended items and thus more representative recommendations.

4 Results

4.1 Original paper reproducibility

The results obtained by reproducing the performance comparison between NGCF and LightGCN at different layers are displayed in Table 3. The numbers for NGCF and LightGCN (reported) methods are directly copied from Table 3 of the LightCGN paper [4]. We add the results of our reproduction of the LightGCN training process as a source of comparison. Both the reported and the reproduced LightGCN results include the percentage of improvement over the NGCF method. The evaluation of all methods is done by employing the Recall@20 and Normalized Discounted Cumulative Gain@20 (NDCG@20) metrics.

The results of the extension to the LastFM dataset are reported in Table 4. The format is similar to the reproduced experiment, with the exception that the results for NGCF and LightGCN (reported) are copied from Table 2 of the paper Graph Trend Filtering Networks for Recommendation [2]. The improvement in percentage over NGCF is calculated by us, since it is not included in the previously mentioned paper.

Dataset		Gowalla		Yelp2018		Amazon-Book	
Layer #	Method	Recall@20	NDCG@20	Recall@20	NDCG@20	Recall@20	NDCG@20
1 Layer	NGCF	15.56%	13.15%	5.43%	4.42%	3.13%	2.41%
	LightGCN (reported)	17.55%(+12.79%)	14.92%(+13.46%)	6.31%(+16.20%)	5.15%(+16.51%)	3.84%(+22.68%)	2.98%(+23.65%)
	LightGCN (reproduced)	16.76% (+10.77%)	14.06% (+10.69%)	5.50%(+10.12%)	4.47%(+10.10%)	3.79%(+21.08%)	2.90%(+20.33%)
2 Layers	NGCF	15.47%	13.07%	5.66%	4.65%	3.30%	2.54%
	LightGCN (reported)	17.77%(+14.84%)	15.24%(+16.60%)	6.22%(+9.89%)	5.04%(+8.38%)	4.11%(+24.54%)	3.15%(+24.02%)
	LightGCN (reproduced)	17.79% (+15%)	15.15% (+15.91%)	5.95%(+5.12%)	4.90%(+8.3%)	4.04%(+22.42%)	3.12%(+22.83%)
3 Layers	NGCF	15.69%	13.27%	5.79%	4.77%	3.37%	2.61%
	LightGCN (reported)	18.23%(+16.19%)	15.55%(+17.18%)	6.39%(+10.38%)	5.25%(+10.06%)	4.10%(+21.66%)	3.18%(+21.84%)
	LightGCN (reproduced)	18.25% (+16.31%)	15.45% (+16.42%)	6.33%(+10.73%)	5.21%(+9%)	4.13%(+22.6%)	3.18%(+21.84%)
4 Layers	NGCF	15.70%	13.27%	5.66%	4.61%	3.44%	2.63%
	LightGCN (reported)	18.30%(+16.56%)	15.50%(+16.80%)	6.49%(+14.58%)	5.30%(+15.02%)	4.06%(+17.92%)	3.13%(+18.92%)
	LightGCN (reproduced)	18.21% (+15.98%)	15.35% (+15.67%)	6.50%(+14.5%)	5.32%(+15%)	4.07%(+17.9%)	3.14%(+18.9%)

Table 3. Performance comparison of NGCF, original LightGCN and reproduced LightGCN over multiple layers, on the Gowalla, Yelp2018 and Amazon-Book datasets.

Dataset		LastFM	
Layer #	Method	Recall@20	NDCG@20
1 Layer	NGCF	7.51%	6.71%
	LightGCN (reported)	8.65%(+15.10%)	7.77%(+15.79%)
	LightGCN (reproduced)	7.58%(+0.9%)	6.80%(+1%)
2 Layers	NGCF	7.66%	6.81%
	LightGCN (reported)	8.74%(+14.09%)	7.88%(+15.71%)
	LightGCN (reproduced)	7.86%(+2%)	7.00%(+2%)
3 Layers	NGCF	7.74%	6.93%
	LightGCN (reported)	8.50%(+9.8%)	7.60%(+9.66%)
	LightGCN (reproduced)	7.80%(+0.7%)	7.00%(+2.5%)
4 Layers	NGCF	7.26%	6.52%
	LightGCN (reported)	8.22%(+13.22%)	7.36%(+12.88%)
	LightGCN (reproduced)	7.54%(+3%)	6.71%(+2%)

Table 4. Performance comparison of NGCF, original LightGCN, and reproduced LightGCN over multiple layers on the LastFM dataset.

4.2 Sampling strategies results

When analyzing the effects of the hard negative sampling on recall and NDCG during the model training, we notice that curves start diverging around epoch 10. The hard negative sampling with a low probability increases the fastest, while the original sampling strategy increases the slowest. However, the values seem to plateau at the same level after approximately 200 epochs. This trend can be seen in Figures 2 and 3 of the Appendix.

Sampling method	Recall@20	NDCG@20	Coverage@20
Original	18.23%	**15.55%**	42.67%
Mixed	18.27% (+0.3%)	15.52% (-0.2%)	**47.91%**(+12.28%)
Hard negative LP	18.27% (+0.3%)	15.47% (-0.5%)	42.4%(-0.63%)
Hard negative HP	18.27% (+0.3%)	15.53% (-0.1%)	46.82%(+9.72%)
Weighted item probability	**18.33%** (+0.5%)	**15.55%**(+0%)	43.94% (+2.97%)

Table 5. Results of different sampling methods applied to the Gowalla dataset with a 3-layer configuration.

Table 5 shows the results of all sampling methods on the Gowalla dataset under a 3-layer configuration. It can be seen that the Weighted item probability sampling leads to the highest Recall@20 and NDCG@20 while the Mixed sampling has the highest Coverage@20. Based on these results, we have determined that the sampling strategies with the most potential are Weighted item probability sampling and Mixed sampling. Therefore, we have tested them on the other 3 datasets to demonstrate their robustness. The results are displayed in Table 6.

Dataset	Method	Recall@20	NDCG@20	Coverage@20
Yelp2018	Original	5.79%	4.42%	38.54 %
	Mixed	6.30%	5.16%	**42.93%**
	Weighted	**6.41%**	**5.25**%	39.01%
Amazon-Book	Original	3.13%	2.41%	37.09%
	Mixed	**4.22%**	**3.27%**	**41.15%**
	Weighted	4.18%	3.22%	37.71%
LastFM	Original	**7.80%**	**7.00%**	55.09%
	Mixed	7.56%	6.74%	**56.57%**
	Weighted	7.77%	6.88%	55.02%

Table 6. Results of Mixed Sampling and Weighted Item Probability Sampling on the Yelp2018, Amazon-Book and LastFM datasets.

In order to further analyze the implications of the sampling methods, the number of seconds per epoch for each method were recorded. The results are

displayed in Table 7. To be noted that these results were given by an NVidia GeForce GTX 1080 Ti GPU.

Sampling method	**Gowalla**	**Yelp2018**	**Amazon-Book**	**LastFM**
Original (C++)	17s	32s	170s	73s
Original (Python)	26s	46s	195s	98s
Mixed	24s	43s	190s	90s
Hard negative (LP&HP)	23s	41s	189s	90s
Weighted item probability	26s	46s	193s	97s

Table 7. Number of seconds per training epoch required by different sampling methods with a 3-layer configuration and a batch size of 1024. For the Amazon-Book dataset, the batch size is 2048.

5 Discussion

Analyzing the results of the reproduction from Table 3, it can be seen that, in most of the cases, the difference between the reported and reproduced LightGCN metrics is smaller than 1%, deeming it insignificant. The only exception is the one-layer configuration, which displays a larger delta between the scores. Since the configurations from Section 4.1.2 of the original paper [4] were used in reproducing all results, we believe that this difference might be explained by the unstable nature of training the model with only one layer. The fact that the reproduced scores become gradually more similar to the reported scores when increasing the number of layers is in support of this argument. Even though a clear discrepancy exists between the results regarding the one layer configuration, the scores of the multiple layer configurations converge, so it can be stated that the results of the original paper are reproducible.

In contrast to the previous results, the metrics in Table 4 display a much larger discrepancy between the reported and reproduced groups. The authors of the Graph Trend Filtering Networks for Recommendation paper [2] state, in section 4.1.4, that the hyper-parameter settings used closely follow the original LightGCN paper for all datasets. However, the LightFM dataset is not mentioned in the original LightGCN paper, leading us to believe that the authors performed a parameter search to achieve their results. Due to time constraints, no parameter search was performed, and the default configuration reported for the Gowalla dataset was used. As a result, the reproduced scores are much lower than the reported ones. Despite this impediment, the metrics show that the LightGCN model still outperforms NGCF in all configurations, demonstrating the robustness of the algorithm on an additional dataset.

Initial experiments confirm the hypothesis about the hard negatives samples: going from a random sampling technique to a hard negative sampling one results in the metrics improving faster, however they fail to surpass the final values of

the random sampling. Nonetheless, we observed that the hard negative sampling strategy increased the catalogue coverage metric and accelerated training. Concerning the plateauing of the metrics, we suggest that the hard negative strategy does help with the model learning at first, but that over the full training schedule of the model, it suffers in terms of data diversity, which hinders the final model performance. Therefore, our work focused on finding the right trade-off between feeding the model hard negative samples and providing a diverse set of samples. In Figures 2 and 3, we saw that the hard negative sampling strategy with the low probability outperformed the high probability one. This indicates that even with a mere 5% inclusion rate of hard negative samples, performance was negatively impacted. We found that introducing hard negative samples at a frequency of 1% allowed the model to access items with greater learning potential while preserving sample diversity.

The results of Table 5 clearly indicate that the method that achieves the best recall is Weighted item probability sampling, while the best coverage is attained by Mixed sampling. Their robustness is further demonstrated by the results in Table 6, which distinctly show that the mixed strategy has the best coverage on all datasets. The highest recall is different for each dataset, although it can be argued that, in most cases, the weighted strategy either achieves the highest score or is short of it by a small margin ($\leq 0.02\%$).

We believe that the weighted strategy is best for recall due to the fact that it attributes higher weights to unpopular items to make them more likely to be positive samples. This, in turn, causes more relevant items to be retrieved, thus increasing recall. In the case of the mixed sampling, it can be said that it benefits from the advantages of selecting positive samples while also limiting the pool of negative samples to achieve the best coverage.

In terms of computational requirements, Table 7 reveals that some sampling methods take longer to execute than others. The largest difference (between Python implementations) can be observed between the Original and Mixed methods on the LastFM database. While the 8 second difference might not seem large at first, it actually implies a contrast of 2.2 hours over the course of 1000 epochs.

6 Individual Contributions

	Extension conceptualization	Implementation (extension and replication)	Report Writing	Poster presentation preparation
Alexandru	33.3	25	40	33.3
Bogdan	33.3	25	40	33.3
Ryan	33.3	50	20	33.3
Total	100%	100%	100%	100%

References

1. Bertin-Mahieux, T., Ellis, D.P., Whitman, B., Lamere, P.: The million song dataset. In: Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011) (2011)
2. Fan, W., Liu, X., Jin, W., Zhao, X., Tang, J., Li, Q.: Graph trend filtering networks for recommendations (2022)
3. He, R., McAuley, J.: Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In: Proceedings of the 25th International Conference on World Wide Web. International World Wide Web Conferences Steering Committee (apr 2016). https://doi.org/10.1145/2872427.2883037, https://doi.org/10.1145%2F2872427.2883037
4. He, X., Deng, K., Wang, X., Li, Y., Zhang, Y., Wang, M.: Lightgcn: Simplifying and powering graph convolution network for recommendation (2020)
5. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks (2017)
6. Liang, D., Charlin, L., McInerney, J., Blei, D.M.: Modeling user exposure in recommendation (2016)
7. Roy, D., Dutta, M.: A systematic review and research perspective on recommender systems. Journal of Big Data **9** (05 2022). https://doi.org/10.1186/s40537-022-00592-5
8. Wang, X., He, X., Wang, M., Feng, F., Chua, T.S.: Neural graph collaborative filtering. In: Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval. ACM (jul 2019). https://doi.org/10.1145/3331184.3331267, https://doi.org/10.1145%2F3331184.3331267

7 Appendix

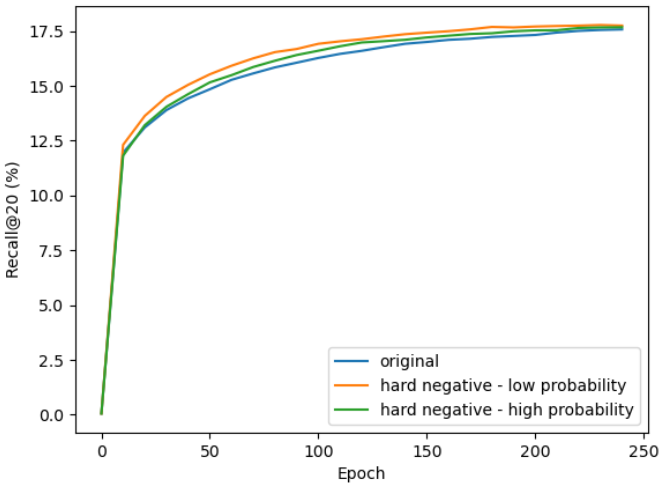

Fig. 2. Recall curves under different sampling strategies - Gowalla dataset, 3 layer configuration

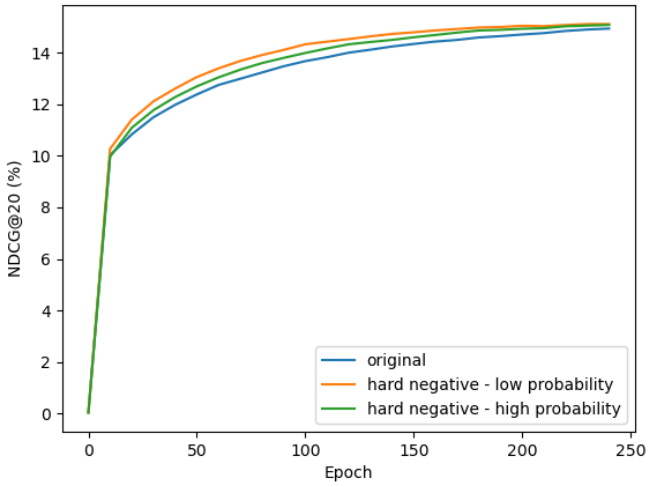

Fig. 3. NDCG curves under different sampling strategies - Gowalla dataset, 3 layer configuration